Input and Interaction

Chapter 3

■ Introduction:

- We now turn to the development of interactive graphics programs.
- Our discussion has three main parts
  - First, we consider the variety of devices available for interaction
  - We then consider client-server networks and client-server graphics
  - Finally we develop a paint program that demonstrates the important features of interactive graphics programming.

1. Interaction

- One of the most important advances in computer technology was enabling users to interact with computer displays
- Ivan Sutherland's *Project Sketchpad* launched the present era of interactive computer graphics
  - Since 1963 when this work was published there have been many advances in both hardware and software, but the viewpoint ideas he introduced still dominate computer graphics.

- OpenGL does not support interaction directly
  - windowing and input functions were left out of the API for portability reasons.
- We can avoid potential difficulties by using a simple library, or toolkit, as we did in Chapter 2.
  - Our use of the GLUT toolkit will enable us to avoid the complexities inherent in the interactions among the windowing system
- Our outline for this chapter looks like this:
  - We start by describing several interactive devices and how to interact with them
  - We then put these devices in the setting of a client-server network.
  - Then, we introduce an API for minimal interaction.
  - Finally, we shall generate sample programs.

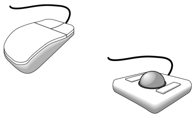
2. Input Devices

- From the physical perspective, each device has properties that make it more suitable for certain tasks than for others.
- There are two primary types of physical devices:
  - Pointing Devices
  - Keyboard Devices

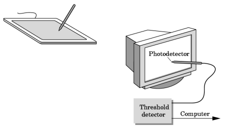
■ 2.1 Physical Input Devices

- The *pointing device* allows the user to indicate a position on the screen,
  - and almost always incorporates one or more buttons
- The *keyboard device* is almost always a physical keyboard,
  - but can be generalized to include any device that returns character codes to a program

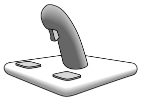
- Pointing devices can be broken into two subcategories
  - relative-position devices



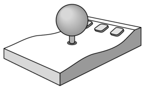
  - absolute- position devices



- Other Pointing Devices
  - Joystick

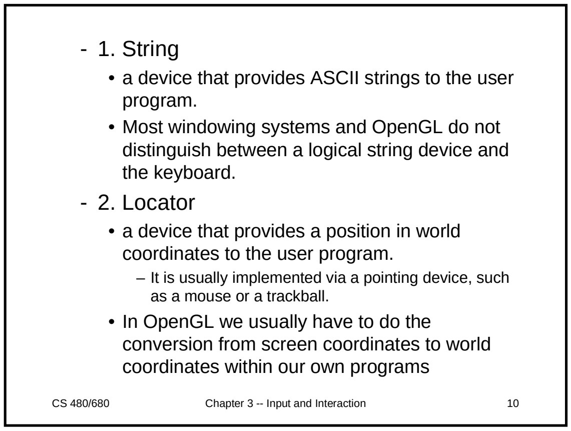


  - SpaceBalls

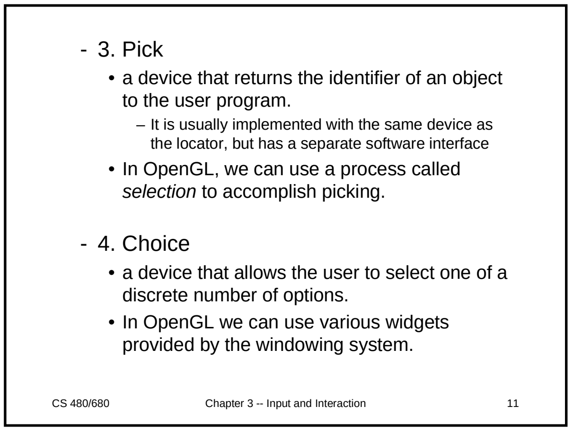


■ 2.2 Logical Devices

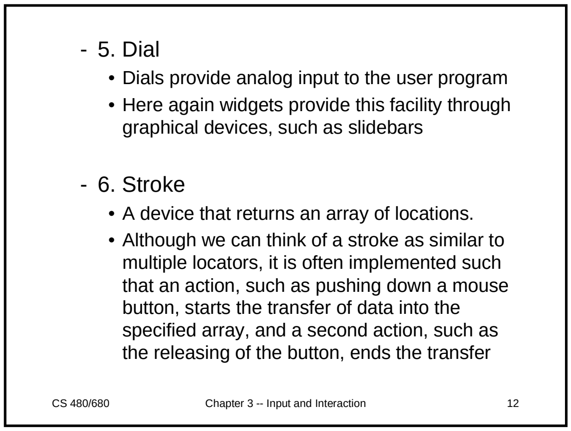
- Two major characteristics describe the logical behavior of an input device
  - What measurements the device returns to the user program
  - When the device returns those measurements.
- Some API's such as PHIGS and GKS consider six classes of logical input devices.
  - OpenGL does not follow this approach, but we will look at these classes briefly and see how OpenGL provides similar functionality



- 1. String
  - a device that provides ASCII strings to the user program.
  - Most windowing systems and OpenGL do not distinguish between a logical string device and the keyboard.
- 2. Locator
  - a device that provides a position in world coordinates to the user program.
    - It is usually implemented via a pointing device, such as a mouse or a trackball.
  - In OpenGL we usually have to do the conversion from screen coordinates to world coordinates within our own programs



- 3. Pick
  - a device that returns the identifier of an object to the user program.
    - It is usually implemented with the same device as the locator, but has a separate software interface
  - In OpenGL, we can use a process called *selection* to accomplish picking.
- 4. Choice
  - a device that allows the user to select one of a discrete number of options.
  - In OpenGL we can use various widgets provided by the windowing system.



- 5. Dial
  - Dials provide analog input to the user program
  - Here again widgets provide this facility through graphical devices, such as slidebars
- 6. Stroke
  - A device that returns an array of locations.
  - Although we can think of a stroke as similar to multiple locators, it is often implemented such that an action, such as pushing down a mouse button, starts the transfer of data into the specified array, and a second action, such as the releasing of the button, ends the transfer

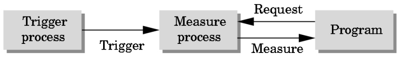
■ 2.3 Measure and Trigger

- The manner by which physical and logical input devices provide input to an application program can be described in terms of two entities:
  - Measure -- what is returned
  - Trigger -- when it is returned
  - Measure can also include status information

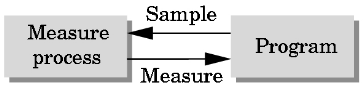
■ 2.4 Input Modes

- We can obtain the measure of a device in three distinct modes.
  - Each mode is defined by the relationship between the measure process and the trigger.
  - The three modes are:
    - Request Mode
    - Sample-Mode
    - Event Mode

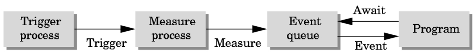
- Request Mode:
  - The measure of the device is not returned to the program until the device is triggered.
  - This mode is standard in non-graphical applications such as typical C programs
    - input is buffered until the trigger is hit
    - for scanf( ) the trigger is the “enter” key.



- Sample Mode:
  - The input is immediate
  - As soon as the function call in the user program is encountered, the measure is returned.
  - Both request mode and sample mode are useful for situations where the program guides the user, but are not useful in applications where the user controls the flow of the program. (Flight Simulator)

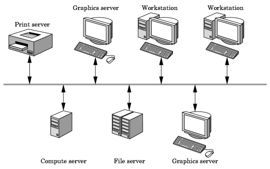


- Event Mode:
  - Each time a device is triggered, an event is generated and placed into an event queue. From here there are two approaches
    - 1) The program can then get events off the queue, and “service” them.
    - 2) associate a “callback” function with a specific type of event and when that event comes to the front of the queue, that function is automatically executed.



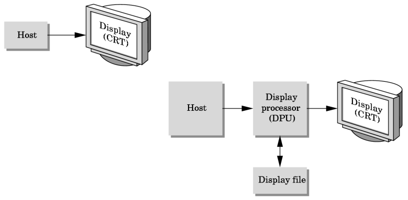
3. Clients and Servers

- Def: Server
  - A Server is something that can perform tasks for clients.
- Clients and Servers can be distributed over a network, or contained entirely within a computational unit.
  - OpenGL programs are clients that use the graphics server



4. Display Lists

- We saw in Chapter 1 the history of graphics hardware



- Today:
  - The display processor has become the graphics server, and the user program has become a client.
  - The issue is not the refresh rate, but the amount of traffic between the client and the server
  - Now we can send graphical entities to a display in one of two ways:
    - immediate mode
    - retained-mode

- Immediate Mode:
  - As soon as the program executes a statement that defines a primitive, that primitive is sent to the server for display and no memory of it is retained.
  - To redisplay it, you must redefine it and resend it
  - For complex objects in highly interactive applications, this process can cause considerable quantity of data to pass from the client to the server

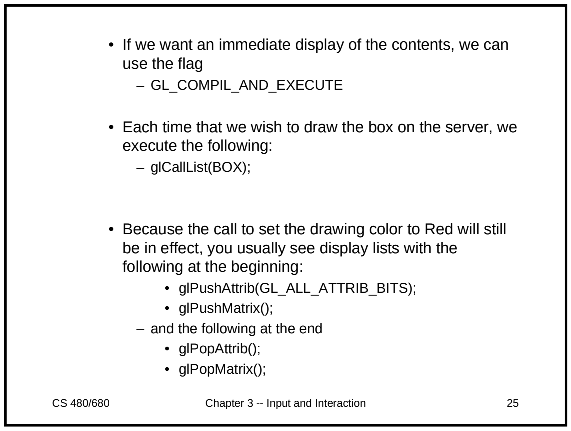
- Retained Mode:
  - We define an object once,
    - then put its description into a display list
  - The display list is stored in the server and redisplayed by a simple function call issued from the client to the server.
  - Advantages:
    - makes good use of hardware
      - (compute and graphics)
  - Disadvantages:
    - memory on the server
    - overhead of creating the lists

■ 4.1 Definition and Execution of Display Lists

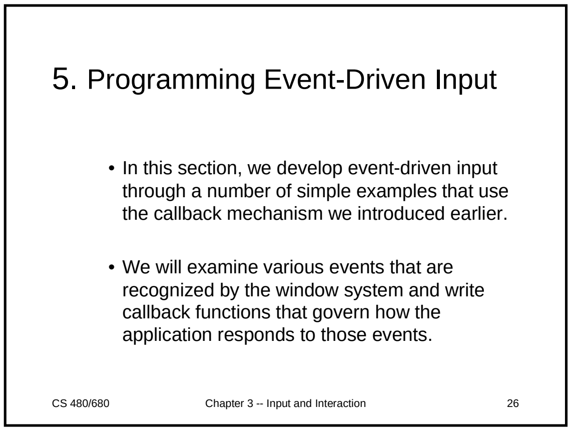
- Display lists have much in common with ordinary files.
  - There must be a mechanism to define/create them as well as and add-to/manipulate them.
  - OpenGL has a small set of functions to manipulate display lists and placed only a few restrictions on display-list contents.
  - We will look at several examples to show their use.

- Display lists are defined similarly to geometric primitives
  - Each display list must have a unique identifier

```
• #define BOX 1
•                                         // GL_COMPILE says
• glNewList(BOX, GL_COMPILE); // send to server
•   glBegin(GL_POLYGON);          // but don't display
•     glColor3f(1.0,0.0,0.0);
•     glVertex2f(-1.0, -1.0);
•     glVertex2f(-1.0, -1.0);
•     glVertex2f(-1.0, -1.0);
•     glVertex2f(-1.0, -1.0);
•   glEnd();
• glEndList();
```

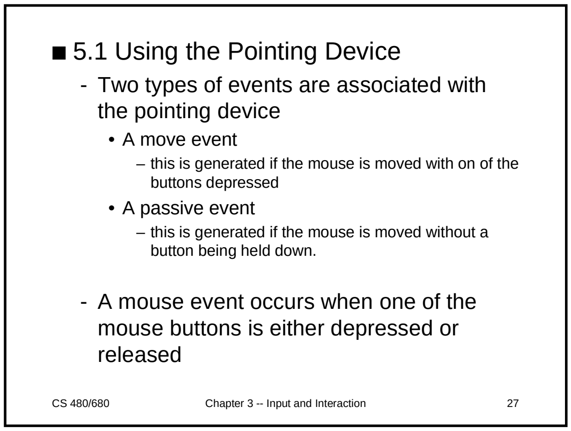


- If we want an immediate display of the contents, we can use the flag
  - GL_COMPIL_AND_EXECUTE
- Each time that we wish to draw the box on the server, we execute the following:
  - glCallList(BOX);
- Because the call to set the drawing color to Red will still be in effect, you usually see display lists with the following at the beginning:
  - glPushAttrib(GL_ALL_ATTRIB_BITS);
  - glPushMatrix();
  - and the following at the end
    - glPopAttrib();
    - glPopMatrix();



5. Programming Event-Driven Input

- In this section, we develop event-driven input through a number of simple examples that use the callback mechanism we introduced earlier.
- We will examine various events that are recognized by the window system and write callback functions that govern how the application responds to those events.



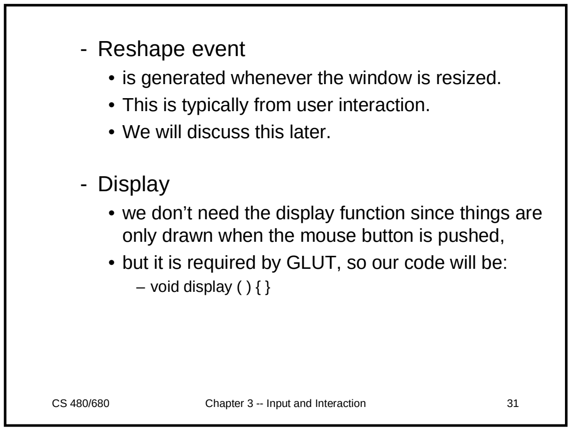
■ 5.1 Using the Pointing Device

- Two types of events are associated with the pointing device
  - A move event
    - this is generated if the mouse is moved with on of the buttons depressed
  - A passive event
    - this is generated if the mouse is moved without a button being held down.
- A mouse event occurs when one of the mouse buttons is either depressed or released

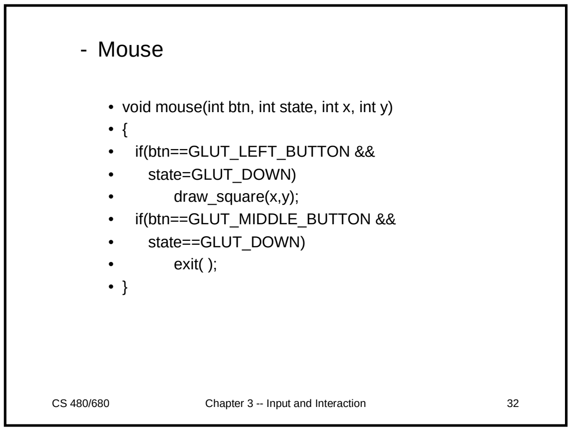
- We specify the mouse callback function in main with the GLUT call
  - glutMouseFunc(mouse_callback_func);
- The mouse callback function must have the form
  - void mouse_callback_func(int button, int state, int x, int y);
- The code for this would look like
  - void mouse_callback_function(int button, int state, int x, int y)
  - {
  -    if(button==GLUT_LEFT_BUTTON &&
  -        state==GLUT_DOWN)
  -            exit();
  - }

- Note:
  - with this code, if any other mouse event had occurred, no response action would occur because no callbacks corresponding to the events has been defined (or registered).
- Our next example
  - is going to draw a box at the mouse location when the left button is pushed
  - and terminate when the middle button is pushed.

- int main(int argc, char **argv)
- {
-    glutInit(&argc, argv);
-    glutInitDisplay(GLUT_SINGLE | GLUT_RGB);
-    glutCreateWindow(“square”);
-    myinit();
-    glutReshapeFunc(myReshape);
-    glutMouseFunc(mouse);
-    glutDisplayFunc(display);
-    glutMainLoop( );
- }

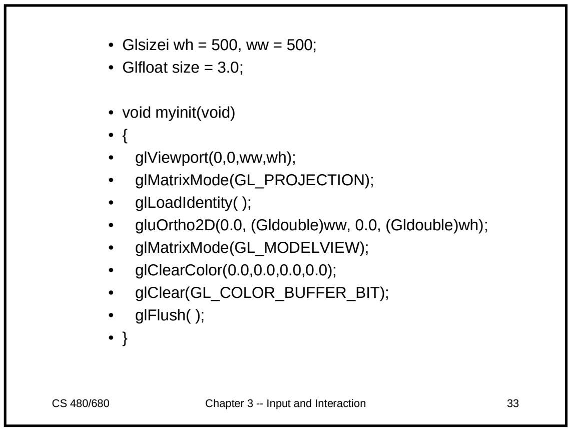


- Reshape event
  - is generated whenever the window is resized.
  - This is typically from user interaction.
  - We will discuss this later.
- Display
  - we don’t need the display function since things are only drawn when the mouse button is pushed,
  - but it is required by GLUT, so our code will be:
    - void display ( ) { }

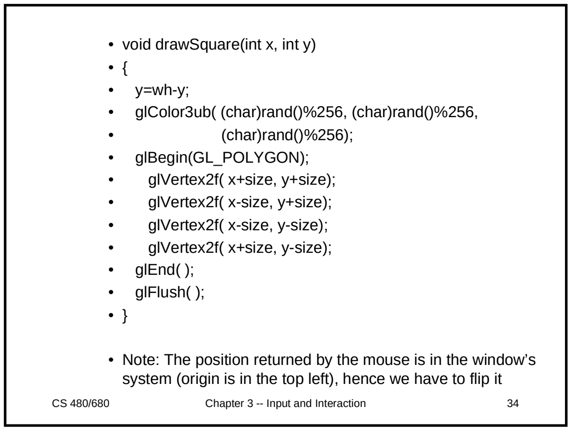


- Mouse

```
• void mouse(int btn, int state, int x, int y)
• {
•   if(btn==GLUT_LEFT_BUTTON &&
•     state=GLUT_DOWN)
•         draw_square(x,y);
•   if(btn==GLUT_MIDDLE_BUTTON &&
•     state==GLUT_DOWN)
•         exit( );
• }
```

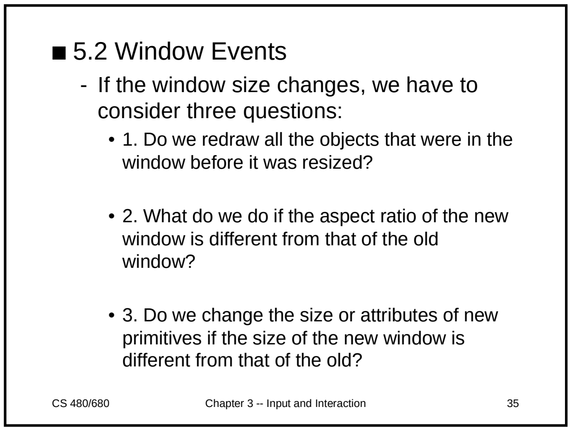


```
• Glsizei wh = 500, ww = 500;
• Glfloat size = 3.0;

• void myinit(void)
• {
•   glViewport(0,0,ww,wh);
•   glMatrixMode(GL_PROJECTION);
•   glLoadIdentity( );
•   gluOrtho2D(0.0, (Gldouble)ww, 0.0, (Gldouble)wh);
•   glMatrixMode(GL_MODELVIEW);
•   glClearColor(0.0,0.0,0.0,0.0);
•   glClear(GL_COLOR_BUFFER_BIT);
•   glFlush( );
• }
```



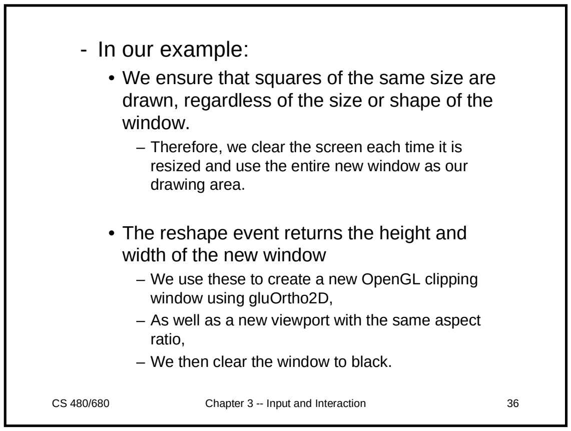
```
• void drawSquare(int x, int y)
• {
•   y=wh-y;
•   glColor3ub( (char)rand()%256, (char)rand()%256,
•                      (char)rand()%256);
•   glBegin(GL_POLYGON);
•      glVertex2f( x+size, y+size);
•      glVertex2f( x-size, y+size);
•      glVertex2f( x-size, y-size);
•      glVertex2f( x+size, y-size);
•   glEnd( );
•   glFlush( );
• }
```

- Note: The position returned by the mouse is in the window's system (origin is in the top left), hence we have to flip it



■ 5.2 Window Events

- If the window size changes, we have to consider three questions:
  - 1. Do we redraw all the objects that were in the window before it was resized?
  - 2. What do we do if the aspect ratio of the new window is different from that of the old window?
  - 3. Do we change the size or attributes of new primitives if the size of the new window is different from that of the old?



- In our example:
  - We ensure that squares of the same size are drawn, regardless of the size or shape of the window.
    - Therefore, we clear the screen each time it is resized and use the entire new window as our drawing area.
  - The reshape event returns the height and width of the new window
    - We use these to create a new OpenGL clipping window using gluOrtho2D,
    - As well as a new viewport with the same aspect ratio,
    - We then clear the window to black.

- void myReshape(Glsizei w, Glsizei h)
- {
-   glMatrixMode(GL_PROJECTION);
-   glLoadIdentity( );
-   gluOrtho2D(0.0, (Gldouble)ww, 0.0, (Gldouble)wh);
-   glMatrixMode(GL_MODELVIEW);
-   glLoadIdentity( );

-   glViewport(0,0,w,h);
-   glClearColor(0.0,0.0,0.0,0.0);
-   glClear(GL_COLOR_BUFFER_BIT);
-   glFlush( );
- }

■ 5.3 Keyboard Events

- Keyboard events are generated when the mouse is in the window and one of the keys is depressed.
  - GLUT ignores key releases

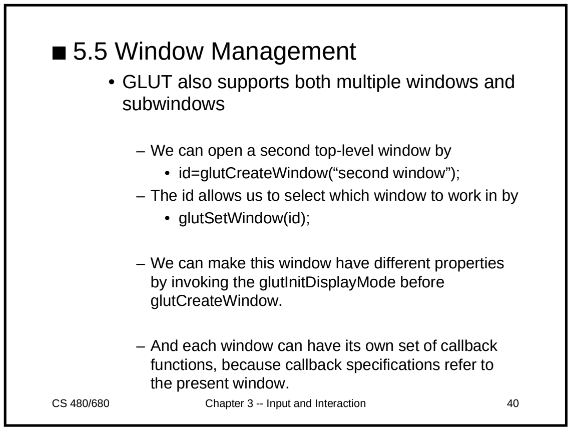
  – All the keyboard callbacks are registered in a single callback function such as:
  - glutKeyboardFunc(keyboard);

  – For example:
  - void keyboard(unsigned char key, int x, int y)
  - {
  -   if(key==‘q’ || key ==‘Q’) exit();
  - }

■ 5.4 The Display and Idle Callbacks

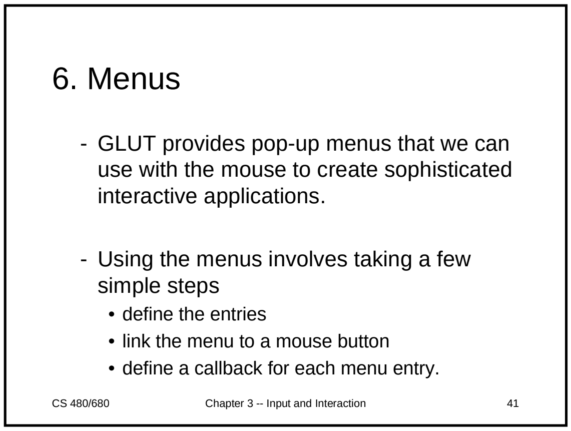
- glutDisplayFunc(display);
  - – display is invoked when GLUT determines that the window should be redisplayed
    - window initially opened,
  - – We can also tell GLUT to call it with
    - glutPostDisplay();

- idle
  - – The idle callback is invoked when there are no other events.
  - – Its default is the null function.



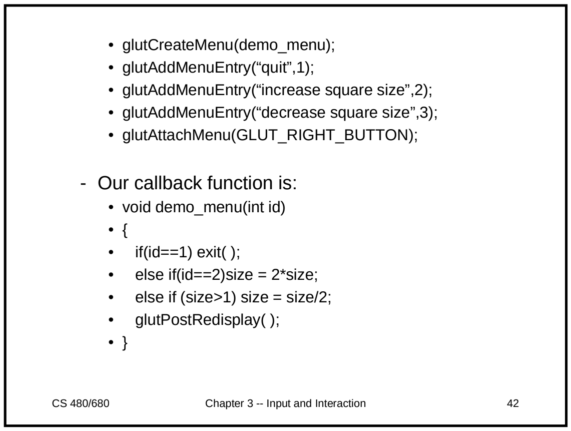
■ 5.5 Window Management

- GLUT also supports both multiple windows and subwindows
  - – We can open a second top-level window by
    - id=glutCreateWindow(“second window”);
  - – The id allows us to select which window to work in by
    - glutSetWindow(id);
  - – We can make this window have different properties by invoking the glutInitDisplayMode before glutCreateWindow.
  - – And each window can have its own set of callback functions, because callback specifications refer to the present window.



6. Menus

- GLUT provides pop-up menus that we can use with the mouse to create sophisticated interactive applications.
- Using the menus involves taking a few simple steps
  - define the entries
  - link the menu to a mouse button
  - define a callback for each menu entry.

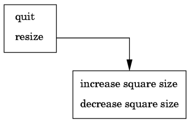


```
• glutCreateMenu(demo_menu);
• glutAddMenuEntry(“quit”,1);
• glutAddMenuEntry(“increase square size”,2);
• glutAddMenuEntry(“decrease square size”,3);
• glutAttachMenu(GLUT_RIGHT_BUTTON);
```

- Our callback function is:

```
• void demo_menu(int id)
• {
•    if(id==1) exit( );
•    else if(id==2)size = 2*size;
•    else if (size>1) size = size/2;
•    glutPostRedisplay( );
• }
```

- GLUT also supports hierarchical menus



  - sub_menu = glutCreateMenu(size_menu);
  - glutAddMenuEntry(“increase square size”,2);
  - glutAddMenuEntry(“decrease square size”,3);

  - glutCreateMenu(top_menu);
  - glutAddMenuEntry(“quit”,1);
  - glutAddSubMenu(“Resize”, sub_menu);
  - glutAttachMenu(GLUTRIGHTBUTTON);

7. Picking

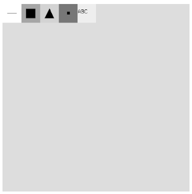
- Picking is an input operation that allows the user to identify an object on the display.

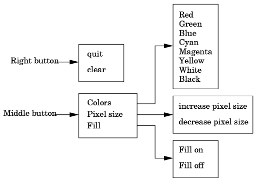
  - Although the action uses the pointing device, the information the user wants is not a position.

  - A pick device is considerably more difficult to implement on a modern system than is a locator
    - The text gives some ideas

8. A Simple Paint Program

- Any paint program should demonstrate most of the following features:
  - ability to work with geometric objects
  - ability to manipulate pixels
  - provide control of attributes (color, ...)
  - include menus for controlling the application
  - behave correctly when the window is moved or resized.





Right button

quit
clear

Middle button

Colors
Pixel size
Fill

Red
Green
Blue
Cyan
Magenta
Yellow
White
Black

increase pixel size
decrease pixel size

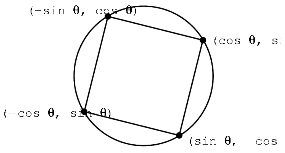
Fill on
Fill off

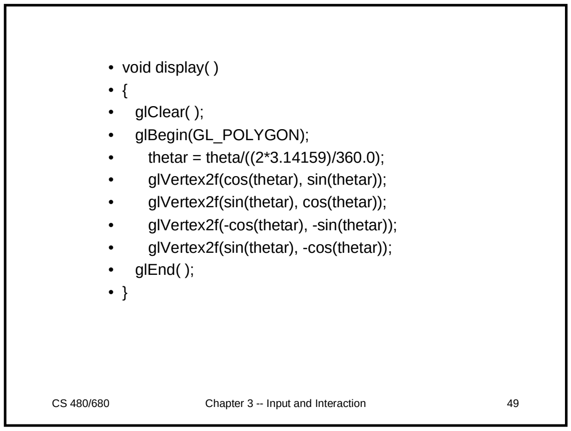
9. Animating Interactive Programs

- So far our programs have been static
  - Once a primitive was placed on the display its image did not change until the screen was cleared.
- Suppose that we want to create a picture in which one or more objects are changing or moving, and thus their images must change.

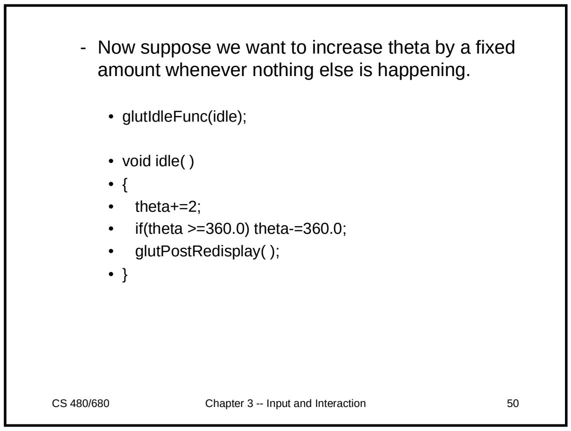
■ 9.1 The Rotating Square

- Consider the rotating two dimensional point where
  - $x = \cos\theta$,
  - $y = \sin\theta$



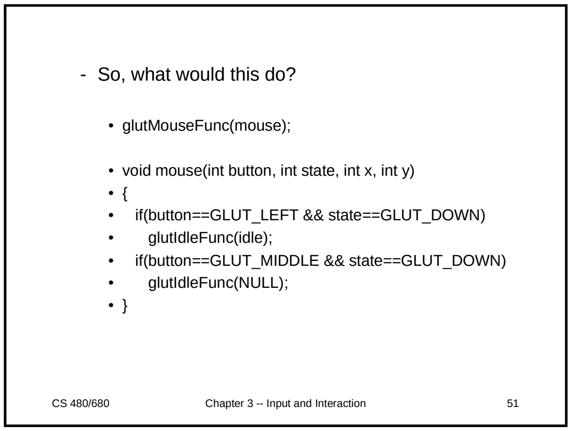


```
• void display( )
• {
•   glClear( );
•   glBegin(GL_POLYGON);
•      thetar = theta/((2*3.14159)/360.0);
•      glVertex2f(cos(thetar), sin(thetar));
•      glVertex2f(sin(thetar), cos(thetar));
•      glVertex2f(-cos(thetar), -sin(thetar));
•      glVertex2f(sin(thetar), -cos(thetar));
•   glEnd( );
• }
```



- Now suppose we want to increase theta by a fixed amount whenever nothing else is happening.

```
• glutIdleFunc(idle);

• void idle( )
• {
•   theta+=2;
•   if(theta >=360.0) theta-=360.0;
•   glutPostRedisplay( );
• }
```



- So, what would this do?

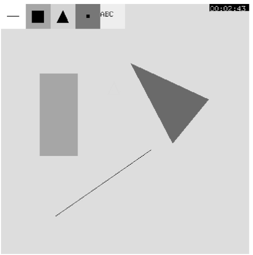
```
• glutMouseFunc(mouse);

• void mouse(int button, int state, int x, int y)
• {
•   if(button==GLUT_LEFT && state==GLUT_DOWN)
•      glutIdleFunc(idle);
•   if(button==GLUT_MIDDLE && state==GLUT_DOWN)
•      glutIdleFunc(NULL);
• }
```

■ 9.2 Double Buffering

- Why do we want double buffering?
- In main we request a double buffered display by:
  - glutInitDisplayMode(GLUT_RGB | GLUT_DOUBLE)
- After we have finished drawing, we call
  - glutSwapBuffers( );

■ 9.3 Other Buffering Problems

- Suppose we want to add an elapsed time clock
  - single buffer causes clock to flicker
  - double buffer causes drawing to flicker
  - answer is to draw the items to both buffers
    - – See Chapter 9 for details



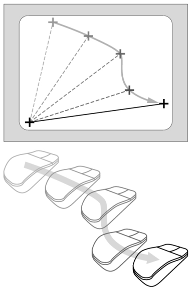
10. Design of Interactive Programs

- A good interactive program should have:
  - A smooth display (no flicker or artifacts of refresh)
  - A variety of interactive devices on the display
  - A variety of methods for entering and displaying information
  - An easy to use interface that does not require substantial effort to learn.
  - Feedback to the user.
  - Tolerance for user errors
  - A design that incorporates consideration of both visual and motor properties of humans.

■ 10.1 Toolkits, Widgets, and the Frame Buffer

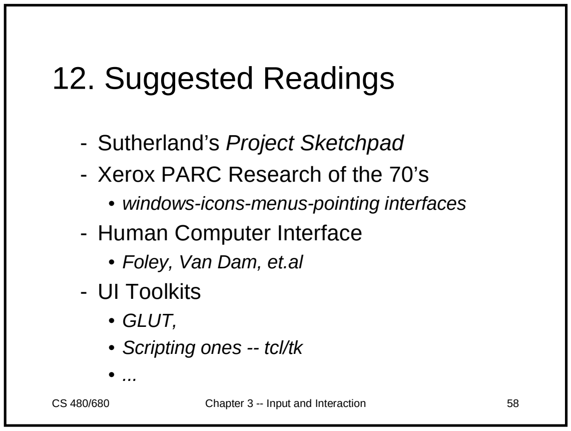
- The paint program used interactive tools, such as pop-up menus that were provided by GLUT.
- There are many more possibilities, such as slide-bars, dials, hot areas on the screen, sound, and icons
  - Usually these tools are supplied with various toolkits, although there is nothing to prevent you from writing your own.
  - Designing our own widget set will be much easier after pixel operations are introduced in Chapter 9
  - most of these operations are described in terms of the contents of the frame buffer
    - bit-block transfer operations (bitblt)

- Rubberbanding is one of those techniques
  - Note that before each new line segment is drawn, the previous line segment must be erased.
  - Usually the rubberbanding begins when the mouse button is depressed and continues until the button is released.
  - Other objects that are drawn interactively with rubberbanding include rectangles and circles.



11. Summary

- In this chapter, we have touched on a number of topics related to interactive computer graphics
  - These interactive aspects make the field exciting and fun.
- We handled simple interaction using GLUT
- We have been heavily influenced by the client-server perspective
  - This will be crucial in Chapter 8
  - The overhead of setting up a program to run in this environment is small. Each program is structured the same.



12. Suggested Readings

- Sutherland's *Project Sketchpad*
- Xerox PARC Research of the 70's
  - *windows-icons-menus-pointing interfaces*
- Human Computer Interface
  - *Foley, Van Dam, et.al*
- UI Toolkits
  - *GLUT,*
  - *Scripting ones -- tcl/tk*
  - *...*